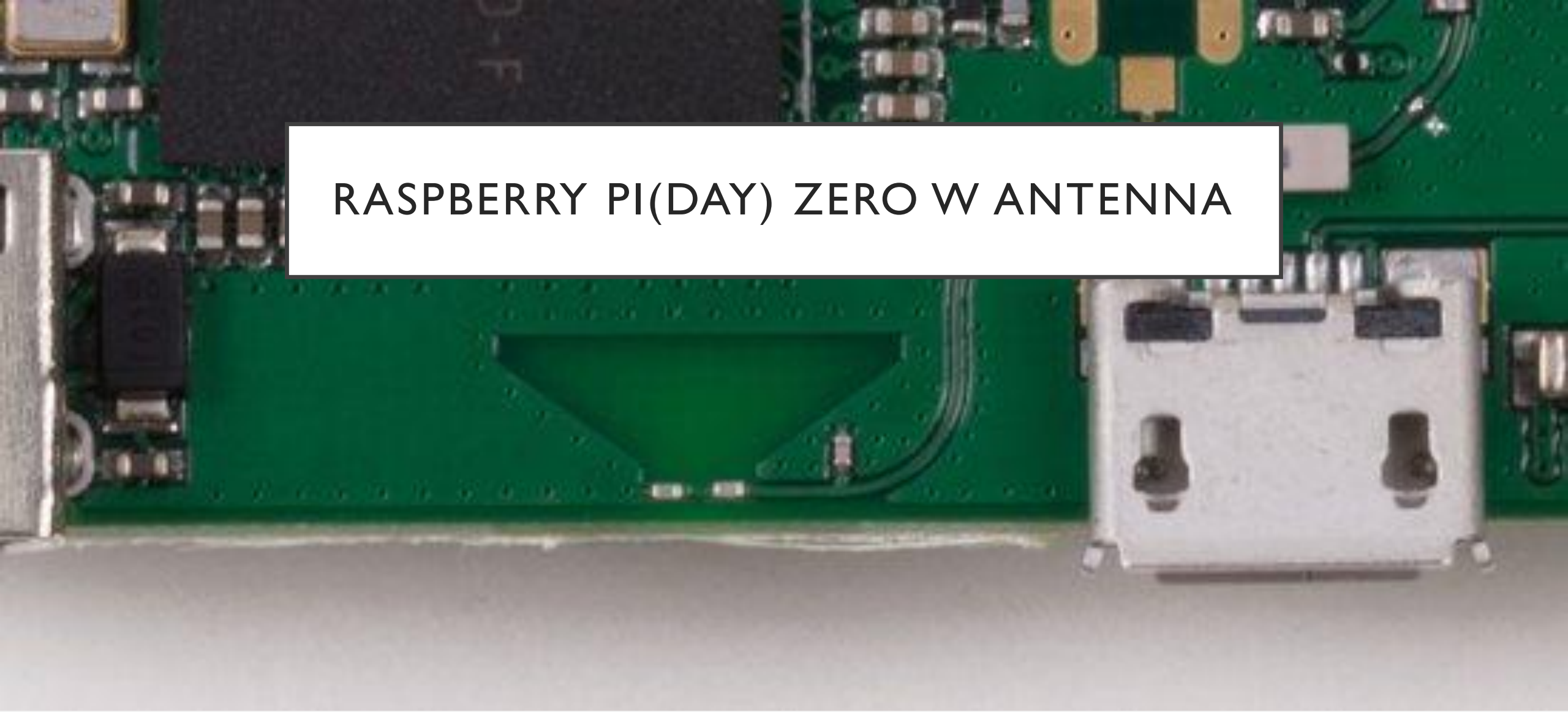
L16: FLASH LAB

ESE516: IoT Edge Computing

Wednesday March 28 2019

Eduardo Garcia – edgarc@seas.upenn.edu



RASPBERRY PI(DAY) ZERO W ANTENNA

- https://www.raspberrypi.org/magpi/pi-zero-w-wireless-antenna-design/
- http://www.proant.se/en/home.htm

TODAY'S LAB GOALS

1. Atmel Studio Solution
    1. Create 2-project solution, including bootloader + app code
    2. Modify the linker and program erasure settings
2. Flash
    1. Connect to your Flash IC using the ASF driver.
    2. Read in the Flash ID to validate the connection.
    3. Read a memory location in flash.
    4. Erase a memory location in flash.
    5. Write a memory location in flash.

BONUS GOALS

If you happen to get through all of the flash…

1. Import and test the ASF **CRC32** driver.
    1. Try running the checksums on a bunch of generated data, writing it to flash, reading it back out, and validating it.
    2. http://asf.atmel.com/docs/3.34.1/samd21/html/group__common__services__crc32.html
2. Import and test the ASF **NVM** driver.
    1. This writes to the SAM D21 non volatile memory. It's what we're going to use to write the application code to memory, while in the bootloader.
    2. http://asf.atmel.com/docs/3.34.1/samd21/html/group__asfdoc__sam0__nvm__group.html

BOOTLOADER REVIEW

PARTITION TABLES

- Remember the caveats of your flash memory!
  - How small of a chunk of memory can you erase?
  - How many bytes can you write at a time?
  - You must erase before writing to a section of memory.
- What do you want to store in your status page?

SAMD21 256kB	0x00000	Bootloader
	0x01F00	Boot status
	0x02000	Application Code
	0x40000	End of memory

External Flash SD CARD	FILE 1	BinaryUpdate
	FILE2	BINARY metadata
	FILE 3	Other Data

STATUS STRUCTURE

- Having a status / header page in memory is a nice way to organize metadata for the firmware binary
- For example, you can include your CRC32 for the entire binary, as well as the size of the file.
- FW_status handles what the current executing image is, the downloaded image, and whether or not I should be writing a new image.
- FW_header keeps track of versioning information – both firmware and hardware. It also holds the CRC for the associated FW image

```c
typedef struct FW_status {
  uint8_t signature[3];     /// Used to determine that partition was initialized
  uint8_t executingImage;   /// Image 1 or 2 in the flash memory
  uint8_t downloadedImage;  /// Image 1 or 2 in the flash memory
  bool writeNewImage;       /// Is a new image ready to be written?
} FW_Status_T;


typedef struct FW_header {
  uint16_t firmwareVersion;
  uint16_t hardwareVersion;
  uint16_t checksum;
} FW_Header_T;
```

```c
/// Read in the boot status
error_code = nvm_read_buffer(BOOT_STATUS_ADDRESS, NVM_buffer_read, NVMCTRL_PAGE_SIZE);
if(error_code != STATUS_OK)
{
  while(1);
}
memcpy(&bootStatus, NVM_buffer_read, sizeof bootStatus);
```

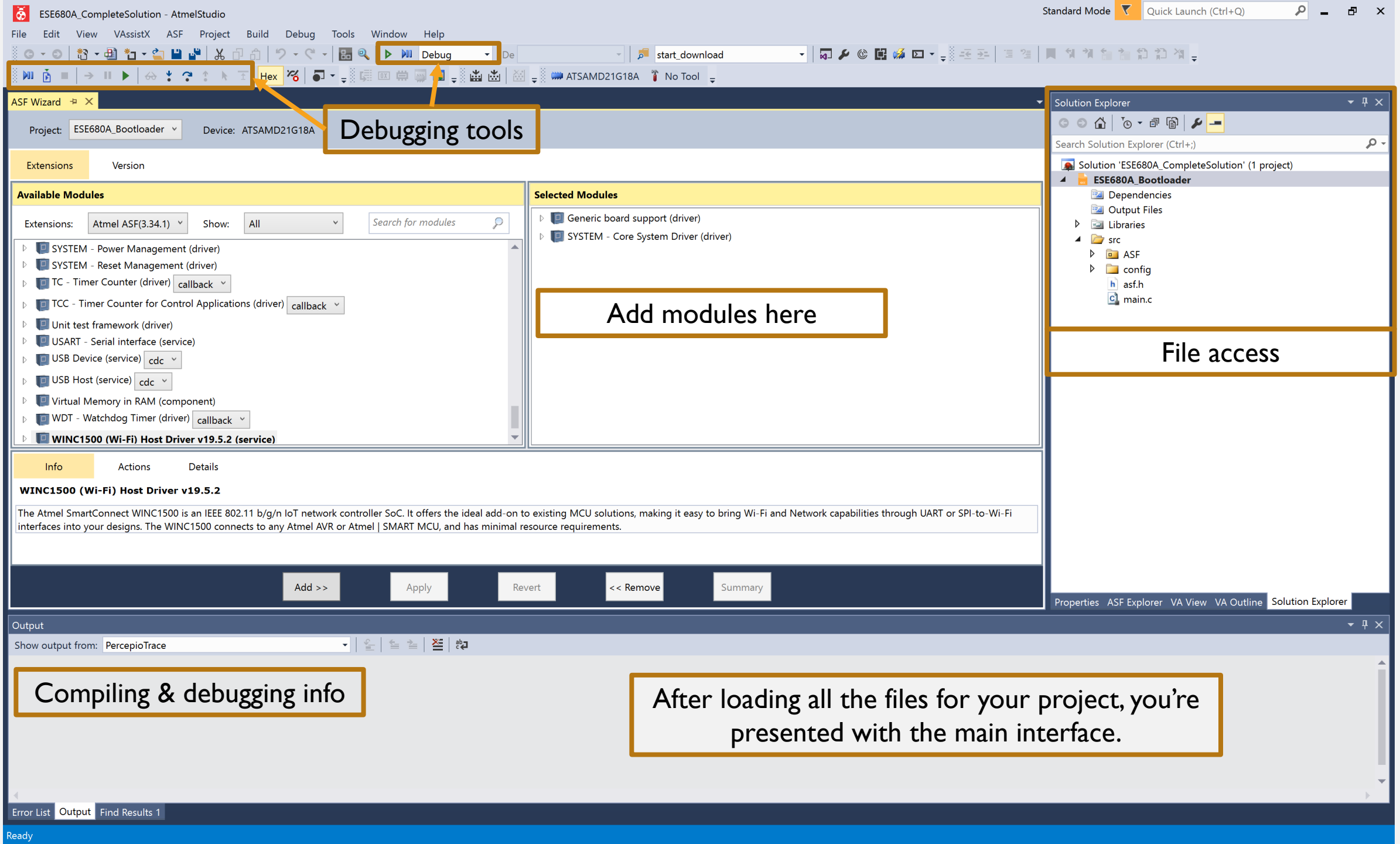
```c
/// If boot status signature is incorrect, write to the first slot in FW
if( bootStatus.signature[0] != 0xAB
||  bootStatus.signature[1] != 0xAC
||  bootStatus.signature[2] != 0xAB)
{
  bootStatus.executingImage   = 2;
}

/// Determine the address to write to in flash
if(bootStatus.executingImage == 1)
{
  flashImageAddress           = FW_IMAGE2_DATA_ADDR;
  bootStatus.downloadedImage  = 2;
}
```

RUNNING APPLICATION CODE

- On boot, the MCU will look to the 0x0000 address – this gives the stack pointer and reset handler for the bootloader
- To transition to the application code, we must point to the application code space – "rebase" the stack pointer & reset vector
- The code on the right handles this functionality
- USB MSC: Page 31-32 has some good references
  - http://www.atmel.com/images/atmel-42352-sam-d21-xpro-usb-host-msc-bootloader_training-manual_an8185.pdf

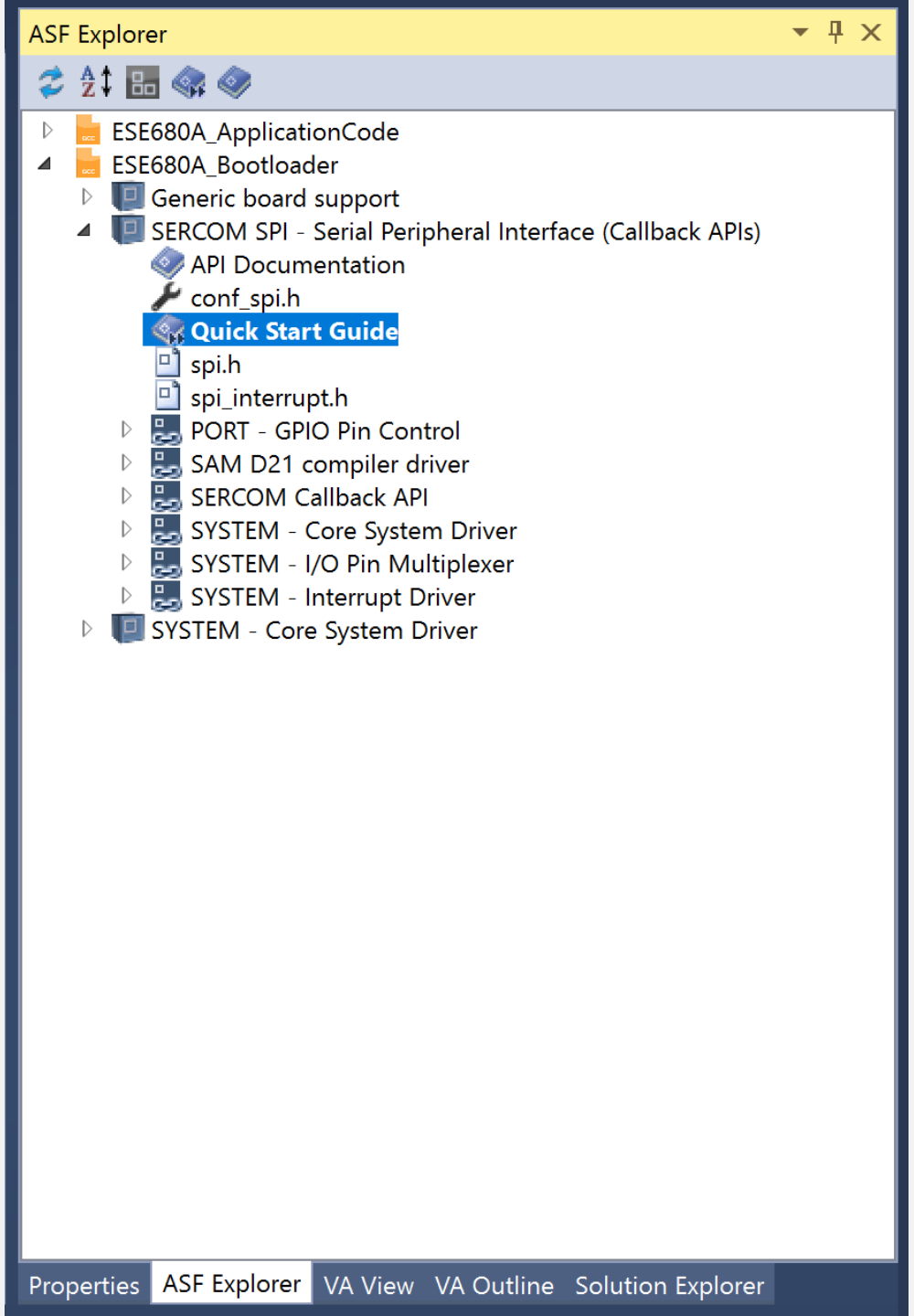
```
/* Pointer to the Application Section */
void (*application_code_entry)(void);

/* Rebase the Stack Pointer */
__set_MSP(*(uint32_t *) APP_START_ADDRESS);

/* Rebase the vector table base address */
SCB->VTOR = ((uint32_t) APP_START_ADDRESS & SCB_VTOR_TBLOFF_Msk);

/* Load the Reset Handler address of the application */
application_code_entry = (void (*)(void))(unsigned *)(*(unsigned *)
(APP_START_ADDRESS + 4));

/* Jump to user Reset Handler in the application */
application_code_entry();
```

ATMEL STUDIO SOLUTION


ESE680A_CompleteSolution - AtmelStudio
Standard Mode
Quick Launch (Ctrl+Q)
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Debug
start_download
Hex
ATSAMD21G18A
No Tool
Debugging tools
ASF Wizard
Project: ESE680A_Bootloader
Device: ATSAMD21G18A
Extensions
Version
Available Modules
Extensions: Atmel ASF(3.34.1)
Show: All
Search for modules
SYSTEM - Power Management (driver)
SYSTEM - Reset Management (driver)
TC - Timer Counter (driver) callback
TCC - Timer Counter for Control Applications (driver) callback
Unit test framework (driver)
USART - Serial interface (service)
USB Device (service) cdc
USB Host (service) cdc
Virtual Memory in RAM (component)
WDT - Watchdog Timer (driver) callback
WINC1500 (Wi-Fi) Host Driver v19.5.2 (service)
Selected Modules
Generic board support (driver)
SYSTEM - Core System Driver (driver)
Add modules here
Info
Actions
Details
WINC1500 (Wi-Fi) Host Driver v19.5.2
The Atmel SmartConnect WINC1500 is an IEEE 802.11 b/g/n IoT network controller SoC. It offers the ideal add-on to existing MCU solutions, making it easy to bring Wi-Fi and Network capabilities through UART or SPI-to-Wi-Fi interfaces into your designs. The WINC1500 connects to any Atmel AVR or Atmel | SMART MCU, and has minimal resource requirements.
Add >>
Apply
Revert
<< Remove
Summary
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ESE680A_CompleteSolution' (1 project)
ESE680A_Bootloader
Dependencies
Output Files
Libraries
src
ASF
config
asf.h
main.c
File access
Properties ASF Explorer VA View VA Outline Solution Explorer
Output
Show output from: PercepioTrace
Compiling & debugging info
After loading all the files for your project, you're presented with the main interface.
Error List Output Find Results 1
Ready

FINDING INFO ON MODULES / DRIVERS

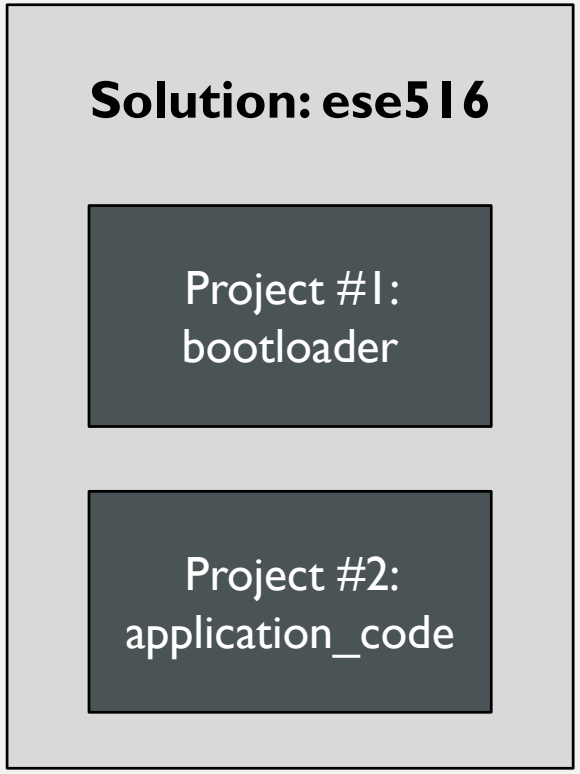
- Atmel has a fair amount of documentation for their drivers.
- In the ASF Explorer panel, you can find:
  - API Documentation: Descriptions of all functions and variables
  - Quick Start Guides (usually): Configuration information and explanations on how to use the code.

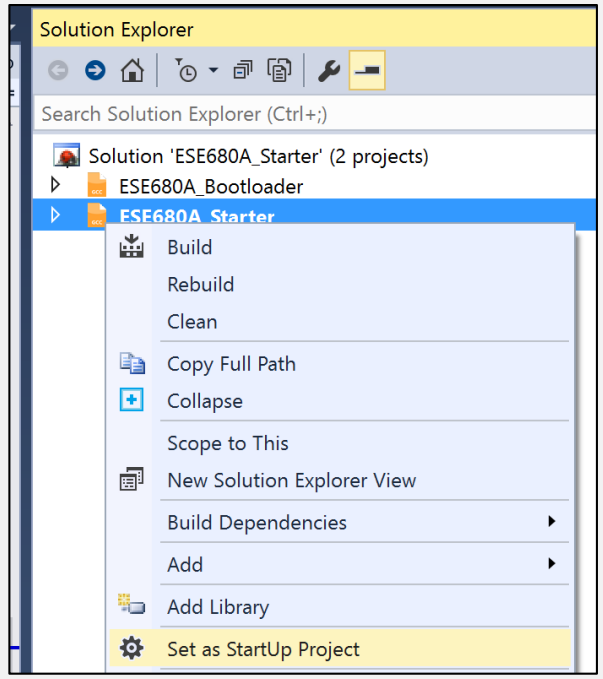


CREATE A 2-PROJECT SOLUTION

SOLUTIONS & PROJECTS

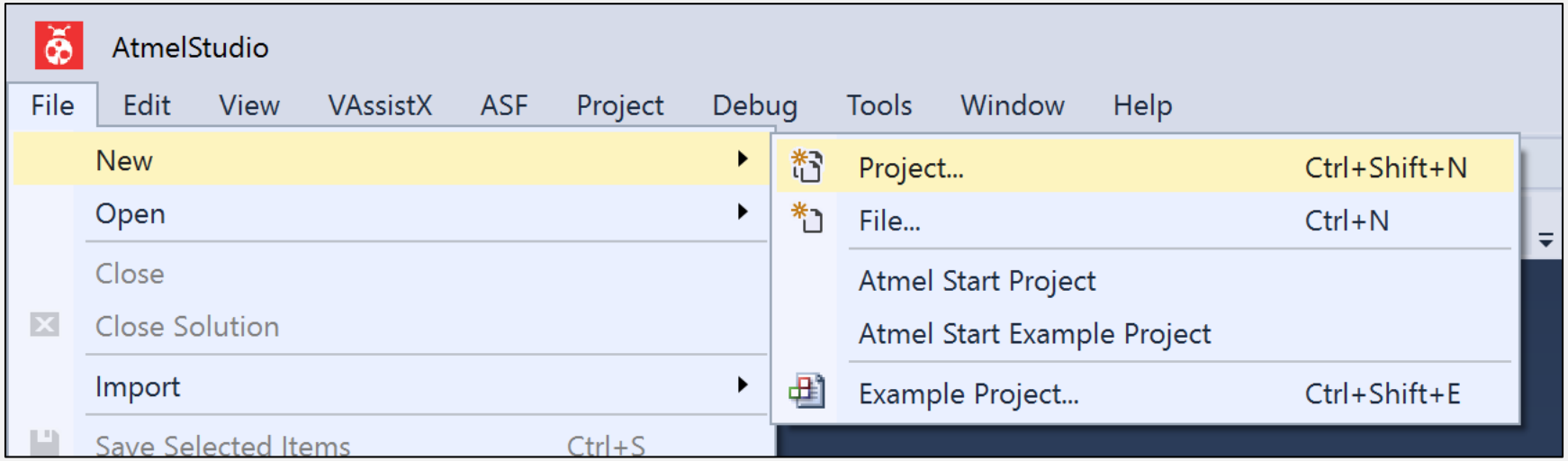
- A project contains multiple source files that compile to a single, flashable binary
- A solution is a collection of related projects – a workspace.
  - You may only target one project for compiling and debugging at a time: **Set as StartUp Project**





BOOTLOADER PROJECT

- You'll want two projects in your solution.
  - One for the bootloader
  - One for the application code
- Start Atmel Studio up, and don't have anything else open.



New Project

Recent

Installed

C/C++

Assembler

AtmelStudio Solution

Sort by: Default

Search Installed Templates (Ctrl+E)

GCC C ASF Board Project C/C++

GCC C Executable Project C/C++

GCC C Static Library Project C/C++

GCC C++ Executable Project C/C++

GCC C++ Static Library Project C/C++

Create project from Arduino sketch C/C++

Type: C/C++

Creates an project for Atmel or User boards using ASF

We'll want to use some of the nice Hardware Abstraction Layer & formatting that comes with using an ASF Board Project.

If you wanted a tight, bare metal design, you could use a standard GCC C project.

Name: ESE680A_Bootloader

Location: c:\users\nmcgill\Documents\Atmel Studio\7.0

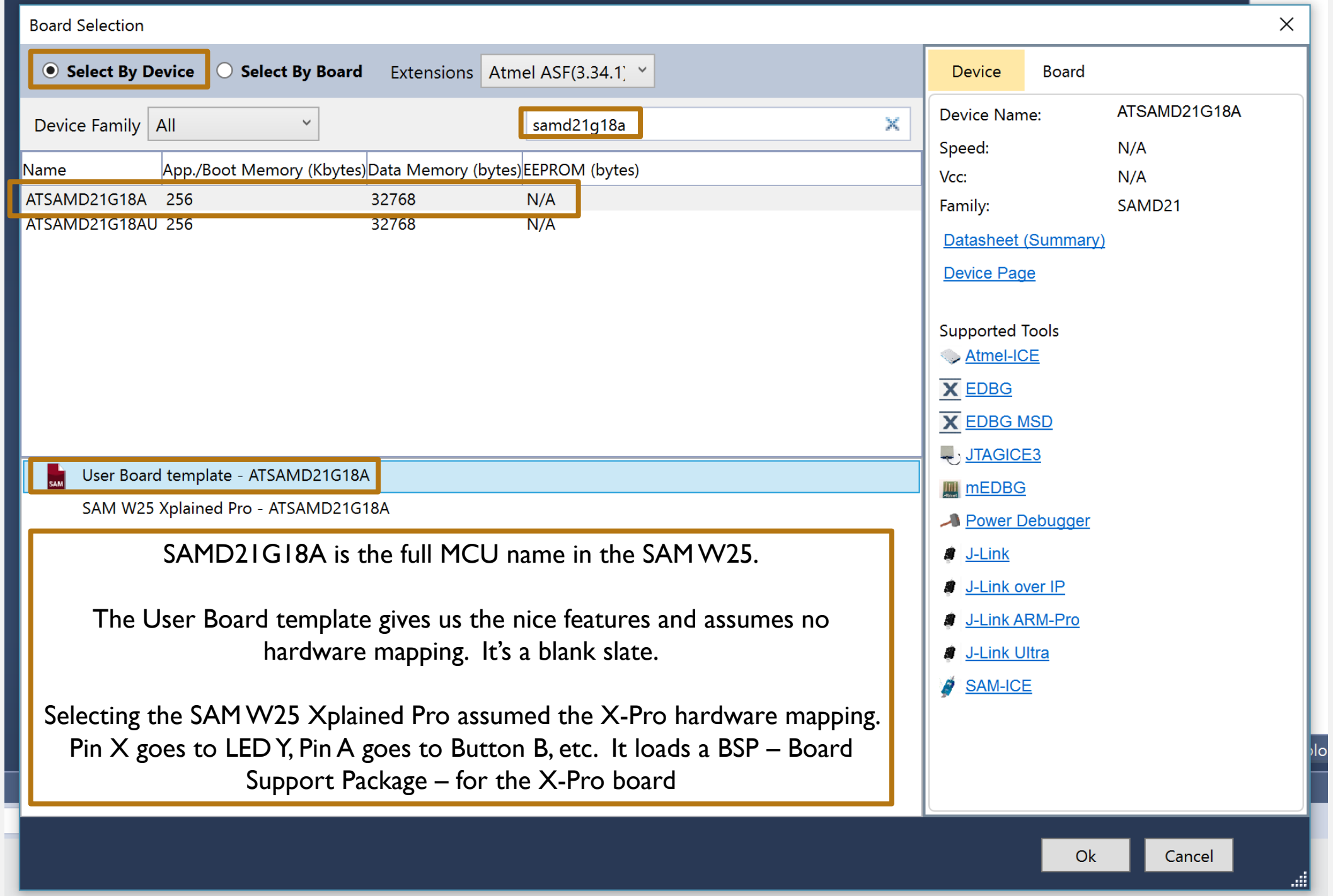
Browse...

Solution name: ESE680A_CompleteSolution

Create directory for solution

OK

Cancel


Board Selection
Select By Device
Select By Board
Extensions
Atmel ASF(3.34.1)
Device Family
All
samd21g18a
Name
App./Boot Memory (Kbytes)
Data Memory (bytes)
EEPROM (bytes)
ATSAMD21G18A 256 32768 N/A
ATSAMD21G18AU 256 32768 N/A
User Board template - ATSAMD21G18A
SAM W25 Xplained Pro - ATSAMD21G18A
SAMD21G18A is the full MCU name in the SAM W25.
The User Board template gives us the nice features and assumes no hardware mapping. It's a blank slate.
Selecting the SAM W25 Xplained Pro assumed the X-Pro hardware mapping. Pin X goes to LED Y, Pin A goes to Button B, etc. It loads a BSP – Board Support Package – for the X-Pro board
Device
Board
Device Name: ATSAMD21G18A
Speed: N/A
Vcc: N/A
Family: SAMD21
Datasheet (Summary)
Device Page
Supported Tools
Atmel-ICE
EDBG
EDBG MSD
JTAGICE3
mEDBG
Power Debugger
J-Link
J-Link over IP
J-Link ARM-Pro
J-Link Ultra
SAM-ICE
Ok
Cancel

New Project

Recent

Installed

C/C++

Assembler

AtmelStudio Solution

Sort by: Default

Search Installed Templates (Ctrl+E)

GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++
Create project from Arduino sketch	C/C++

Type: C/C++

Creates an project for Atmel or User boards using ASF

Name: ESE680A_ApplicationCode

Location: C:\Users\nmcgill\Documents\Atmel Studio\7.0\ESE680A_CompleteSolution

Browse...

Solution: Add to solution

Solution name: ESE680A_ApplicationCode

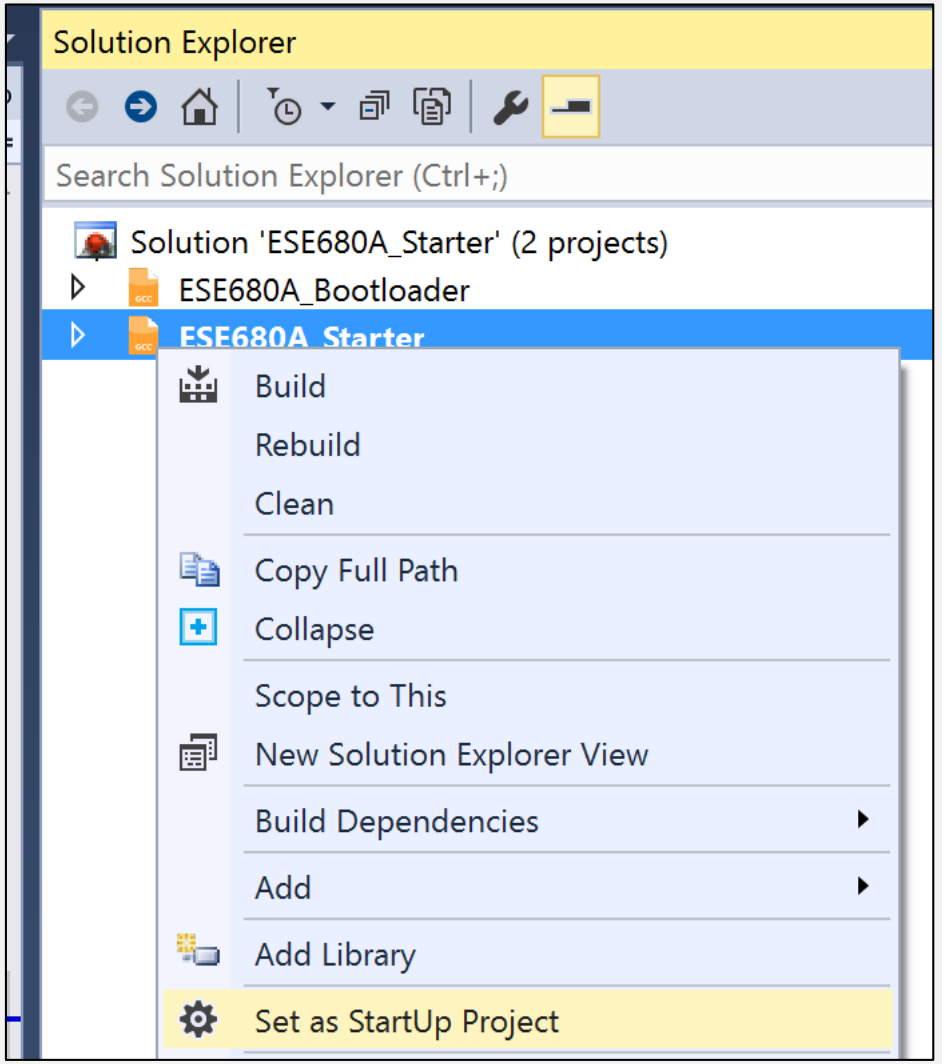
Create directory for solution

OK

Cancel

SPECIFYING A PROJECT IN A MULTI-PROJECT SOLUTION

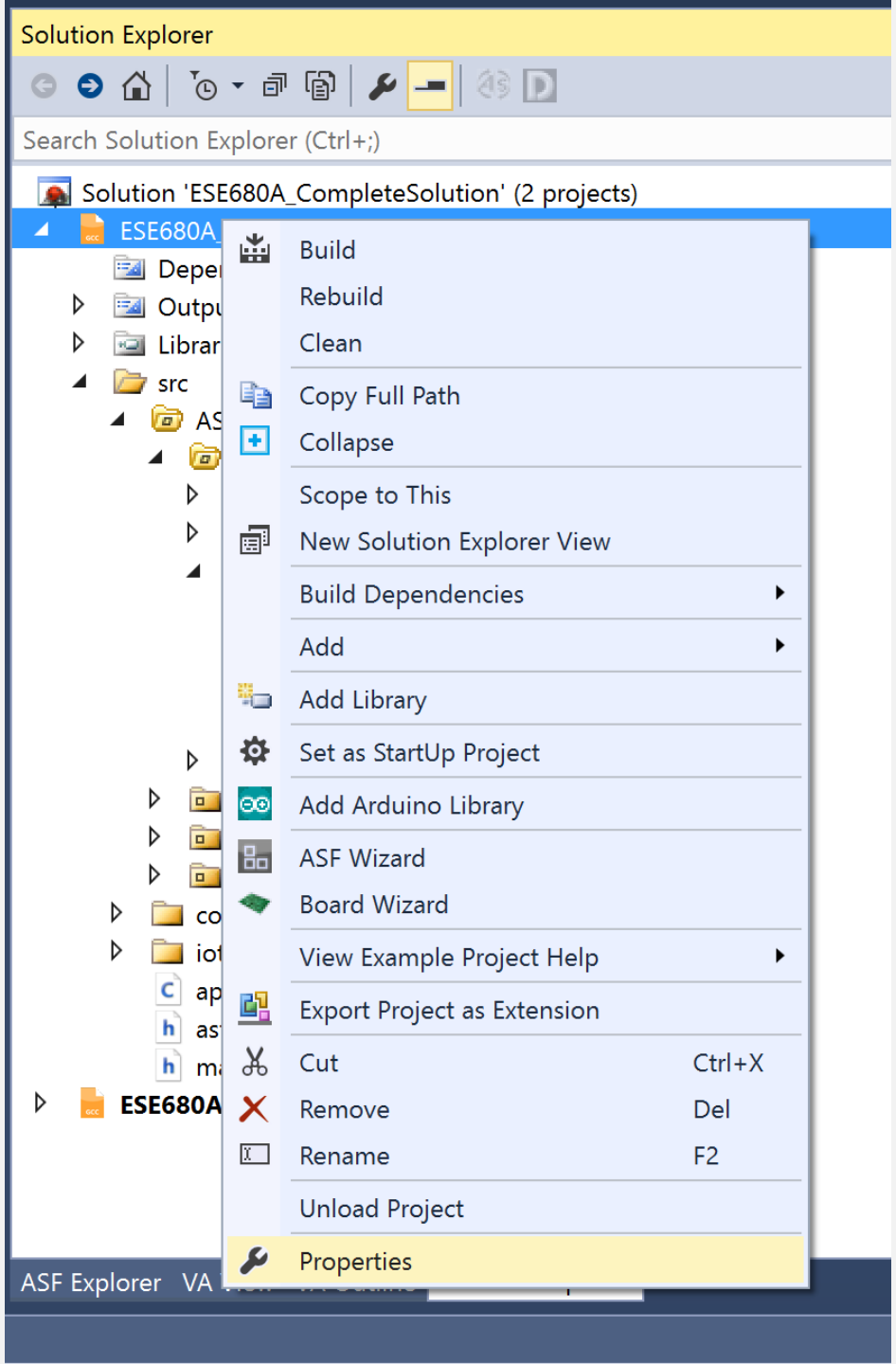
- Once you have more than one project in a solution, Atmel Studio's compiling / debugging controls will only load one project at a time.
- How do you switch between the two?
  - Find the **Solution Explorer** panel.
  - Right-click the desired project, and click **Set as StartUp Project**
  - Debug away!



PROJECT SETTINGS

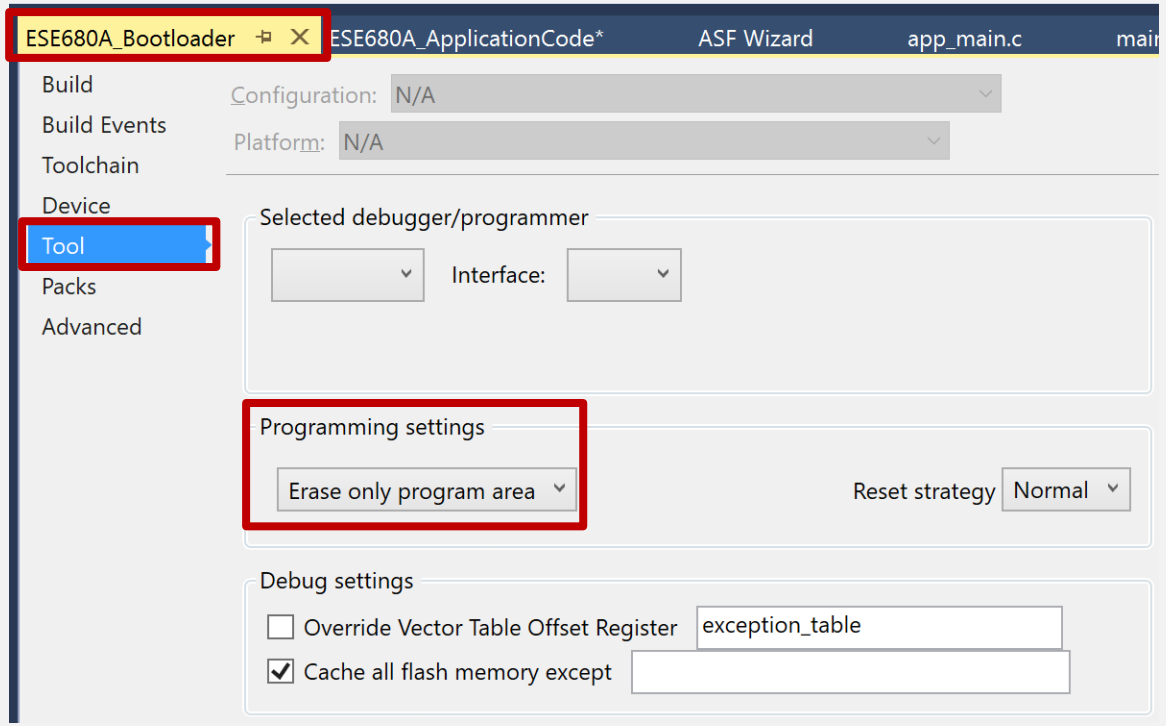
PROJECT PROPERTIES

- Project Properties includes:
  - Build settings (optimizations)
  - Debugging tool support
  - Pre & post build events (do you want to run scripts? Build in an SVN revision number?)



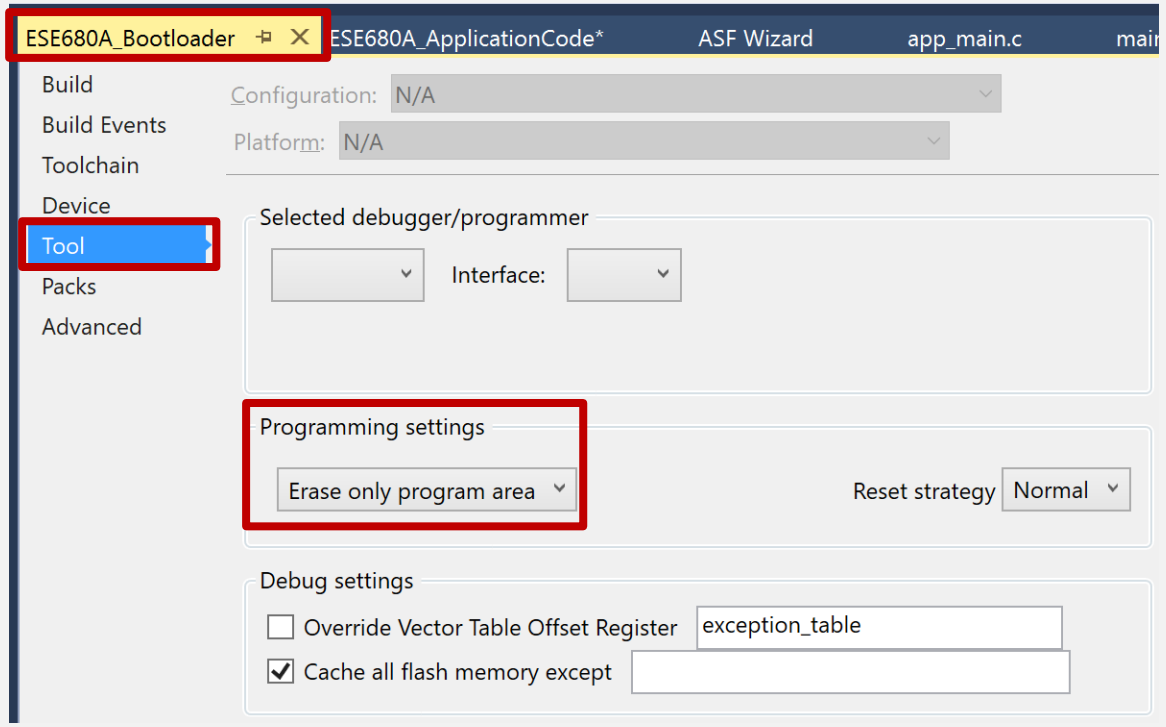
BOOTLOADER CODE: SET PROGRAM ERASURE

- If we want to load the bootloader and application code from Atmel Studio, we want to make sure we're not erasing the wrong parts of code.
- You may want to do differently, but you can set the **Programming settings** to only erase the program area for the bootloader.
- This way, you can update your bootloader code without erasing your app code.
- In **Project Properties**, find the **Tool** pane on the left hand side to set this.



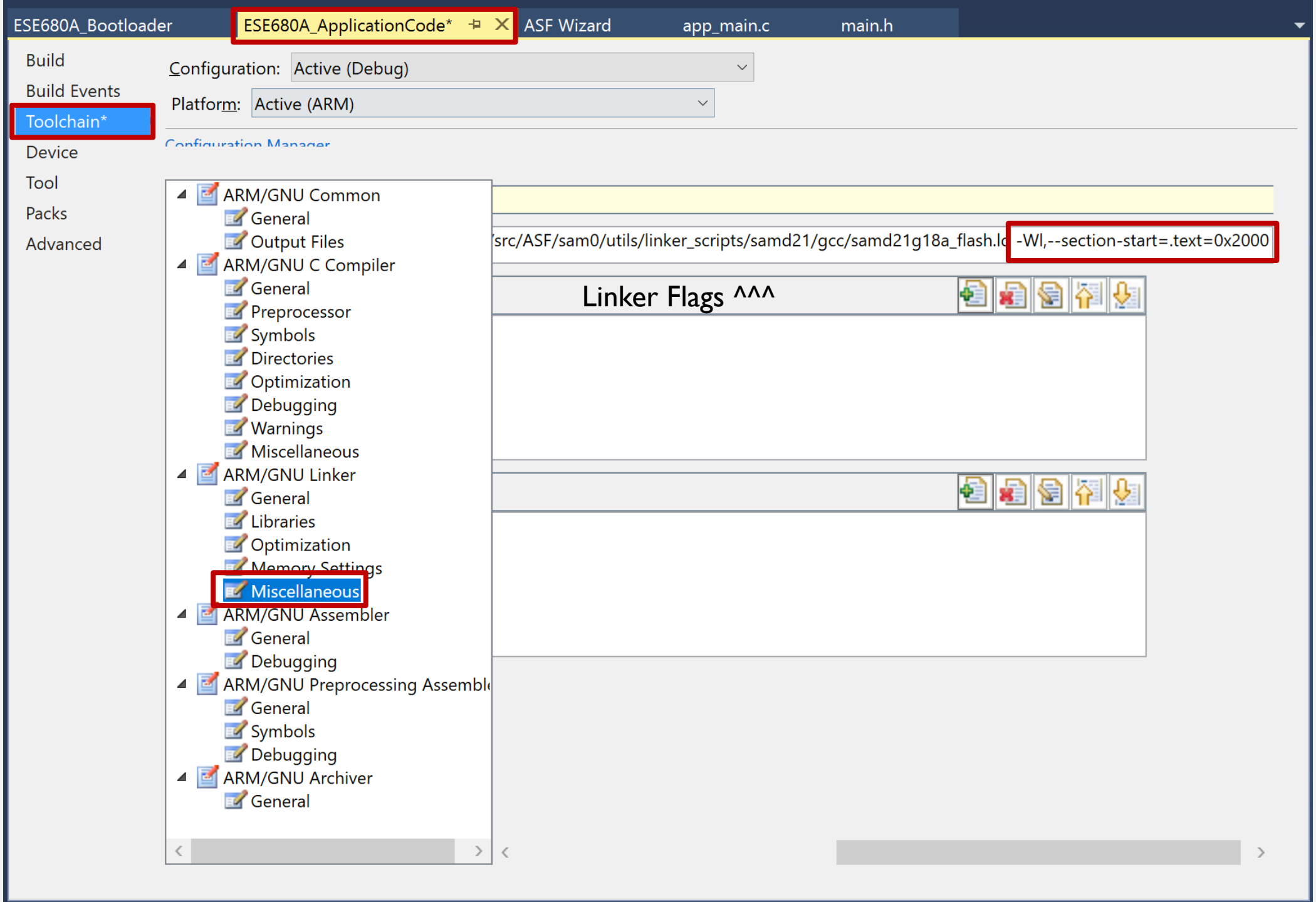
APP CODE: SET PROGRAM ERASURE

- Same thing as the bootloader. When you load your app code, you don't want to erase the bootloader each time.
- In **Project Properties**, find the **Tool** pane on the left hand side to set this.



APP CODE: SET LINKER OFFSET

- Our application code will not live at 0x0000, the default address. It will be living at **0x2000** into memory.
- We need to set this appropriately in our compiling settings so the code knows where it lives. This is handled by the **linker offset**.
- You can find it in **Toolchain > Miscellaneous > Linker Flags**
- Add " **-Wl,--section-start=.text=0x2000**" to the end of your linker flags. Mine looks like:
  - -Wl,--entry=Reset_Handler -Wl,--cref -mthumb -T../src/ASF/sam0/utils/linker_scripts/samd21/gcc/samd21g18a_flash.ld **-Wl,--section-start=.text=0x2000**
- Learn more about Linkers: https://www.microchip.com/webdoc/GUID-ECD8A826-B1DA-44FC-BE0B-5A53418A47BD/index.html?GUID-5DAF6D8F-E607-4966-B2F8-47636FF42957


ESE680A_Bootloader
ESE680A_ApplicationCode*
ASF Wizard
app_main.c
main.h
Build
Build Events
Toolchain*
Device
Tool
Packs
Advanced
Configuration: Active (Debug)
Platform: Active (ARM)
ARM/GNU Common
General
Output Files
ARM/GNU C Compiler
General
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ARM/GNU Linker
General
Libraries
Optimization
Memory Settings
Miscellaneous
ARM/GNU Assembler
General
Debugging
ARM/GNU Preprocessing Assembl
General
Symbols
Debugging
ARM/GNU Archiver
General
src/ASF/sam0/utils/linker_scripts/samd21/gcc/samd21g18a_flash.ld
-Wl,--section-start=.text=0x2000
Linker Flags ^^^

TIPS

NVM ASF DRIVER

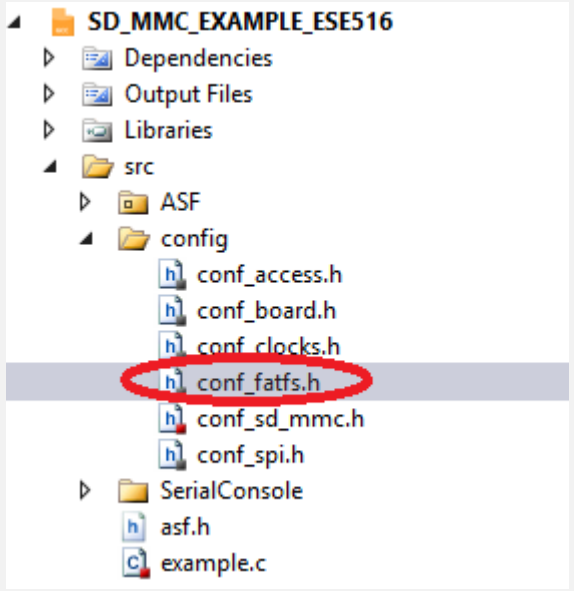
- ASF has a driver (NVM) that has an API to write to the MCU's flash.
- Try the example! It does reads and write – just what you need right? http://asf.atmel.com/docs/latest/samd21/html/asfdoc_sam0_nvm_basic_use_case.html

- Read the documentation! http://asf.atmel.com/docs/latest/samd21/html/group__asfdoc__sam0__nvm__group.html#asfdoc_sam0_nvm_examples

FATFS

FATFS API

- Use the starter code for A6 to have a ready-to-go project with the SD Card and FatFS
- See "http://elm-chan.org/fsw/ff/00index_e.html " for a list of the PAI present
  - The example uses f_gets and f_puts which deal with text. If you want to read and write raw bytes, use f_read and f_write.
- Not all options are turned on by default – if you need an extra function, please check "**conf_fatfs.h**" to turn the option you need on.



JUMPING TO MAIN APPLICATION

RUNNING APPLICATION CODE

- On boot, the MCU will look to the 0x0000 address – this gives the stack pointer and reset handler for the bootloader
- To transition to the application code, we must point to the application code space – "rebase" the stack pointer & reset vector
- The code on the right handles this functionality
- USB MSC: Page 31-32 has some good references
  - http://www.atmel.com/images/atmel-42352-sam-d21-xpro-usb-host-msc-bootloader_training-manual_an8185.pdf

```
/* Pointer to the Application Section */
void (*application_code_entry)(void);

/* Rebase the Stack Pointer */
__set_MSP(*(uint32_t *) APP_START_ADDRESS);

/* Rebase the vector table base address */
SCB->VTOR = ((uint32_t) APP_START_ADDRESS & SCB_VTOR_TBLOFF_Msk);

/* Load the Reset Handler address of the application */
application_code_entry = (void (*)(void))(unsigned *)(*(unsigned *)
(APP_START_ADDRESS + 4));

/* Jump to user Reset Handler in the application */
application_code_entry();
```

```c
#define APP_START_ADDRESS  ((uint32_t)0x2000) //Must be address of start of main application

/// Main application reset vector address
#define APP_START_RESET_VEC_ADDRESS (APP_START_ADDRESS+(uint32_t)0x04)

/**********************************************************************//**
* function      jumpToApplication()
* @brief        Jumps to main application
* @details      Detailed Description
* @note         Add a note
* @param[in]    arg1 Input parameter description
* @param[out]   arg2 Output parameter description
* @return       Description of return value
**********************************************************************/
static void jumpToApplication(void)
{
    /// Function pointer to application section
    void (*applicationCodeEntry)(void);

    /// Rebase stack pointer
    __set_MSP(*(uint32_t *) APP_START_ADDRESS);

    /// Rebase vector table
    SCB->VTOR = ((uint32_t) APP_START_ADDRESS & SCB_VTOR_TBLOFF_Msk);

    /// Set pointer to application section
    applicationCodeEntry =
        (void (*)(void))(unsigned *)(*(unsigned *)(APP_START_RESET_VEC_ADDRESS));

    /// Jump to application
    applicationCodeEntry();
}
```